

Seven Fundamentals of Mission-Critical Service Testing

Robert D. Schneider
November 20, 2009

robert.schneider@think88.com

Agenda

- Introduction
- About Think88
- How We Arrived at this Methodology
- Why Service Testing is Different
- Seven Fundamentals of Mission-Critical Service Testing
- Questions & Answers

About Think88

- Specialized consultancy
 - Silicon Valley-based
 - Global alliances
- Key practices
 - Service Oriented Architecture (SOA)
 - Architecture, design, testing
 - Training & certifications
 - Governance
 - Cloud Computing
 - Training & consulting

Seven Fundamentals of Mission-Critical Service Testing

- How did we come up with this list?
 - Interviews with organizations in varying industries, all deploying SOA & services
 - Finance
 - Airline & transport
 - Government & military
 - Technology
 - Consultancies
 - Peer review
 - Vendor feedback

Seven Fundamentals of Mission-Critical Service Testing

- Regardless of industry, we found a number of consistent trends
 - Trying to do too much with too little
 - Time pressure
 - Cost constraints
 - Lack of a formalized methodology
 - Cutting corners
 - Not running thorough tests
 - Testing with very small data sets
 - Very limited results analysis
 - IT management often doesn't realize that service testing is a wholly different proposition
 - Organizations still using UI-driven testing guidelines for services
- All of these factors led to the creation of these guidelines

What are Services?

- Web services are not the only choice when delivering a service
- It's important to remember that a software service can be implemented in many ways:
 - POJO (Plain old Java object)
 - EJB (Enterprise Java bean)
 - SOAP-based Web service
 - REST-based service
 - Other alternatives
- Service Oriented Architecture (SOA) is an entirely different discussion

Service-Oriented Architecture

SOA is essentially a distinct technology architecture established in support of service-oriented solutions and therefore shaped by the demands and requirements of applying service-orientation.

The fundamental characteristics of SOA are:

- business-driven
- vendor-agnostic
- enterprise-centric
- composition-centric

Source: SOAGlossary.com

What's Different about Testing Services?

- For GUI-based applications, users interaction is fairly constrained
- But *anyone* can send *anything* to a service
 - From anywhere
- This necessitates much more rigorous testing
 - Of business logic
 - Of data
 - Of exception handling
 - Of security

What's Different about Testing Services?

- With SOA, services play a much more vital role than any given siloed application
 - They often serve as the only way to access key enterprise business logic
- Performance testing becomes much more important
 - Of the service itself
 - Of compositions that use the service
- SLA compliance will often make-or-break a service

What's Different about Testing Services?

- For a siloed application, a governance failure inconveniences a subset of the user community
- On the other hand, a SOA governance failure can jeopardize the entire enterprise
 - The service might be used by a whole suite of solutions
- Thus, governance-related tests are especially important for services

Seven Fundamentals of Mission-Critical Service Testing

1. Thoroughly Test Your Services
2. Test Using Large Amounts of Realistic Data
3. Make Sure Your Services Are Secure
4. Get the Most Productivity from Your Developers and Testers
5. Fully Track Your Test Results
6. Test Your Services Under Anticipated Loads
7. Make Sure You Govern Your Services

Important: while we illustrate examples with soapUI Pro, the principles themselves are vendor-neutral

#1: Thoroughly Test Your Services

Thoroughly Test Your Services

- Problems
 - Far too many developers and testers take a ‘hello world’ approach to testing a service
 - i.e. if it returns some data, it passes
 - This isn’t sufficient for the real world
 - Even when tests are more comprehensive, they often aren’t tied to a requirements management system
 - Complex Web services have complex messages, yet testing often only covers a fraction of these messages

Thoroughly Test Your Services

- Implications
 - Failure to truly exercise application logic
 - Incomplete error checking
 - No test repeatability
 - Difficult to pinpoint service issues

Thoroughly Test Your Services

- Best Practices
 - Make sure that all messages are covered
 - Make sure that all elements are covered
 - Write test cases even if the services aren't finished yet
 - Use a QA management system
 - Test corner and boundary conditions
 - Test complex messages
 - Including attachments
 - Apply assertions to results
 - Test even if there isn't a contract

Viewing Coverage Metrics

The screenshot displays a testing tool interface with a tree view on the left and a message content pane on the right. The tree view shows a hierarchy of elements with their respective contract coverage percentages and counts. The 'Request' element under the 'Message' node is highlighted in blue.

Element	Contract Coverage	Count
110.90.12	5% (0%)	11/241
OrderRatingServiceSoapBind	5% (0%)	11/241
getCommOrderRate	0% (0%)	0/104
getCommOrderServiceSt	0% (0%)	0/26
getDistrOrderRate	10% (0%)	11/111
OrderRatingServiceSoapBind	5% (0%)	11/241
getCommOrderRate	0% (0%)	0/104
getCommOrderServiceSt	0% (0%)	0/26
getDistrOrderRate	10% (0%)	11/111
Message	10% (0%)	11/111
Request	24% (0%)	8/34
Response	4% (0%)	3/77

The message content pane shows a SOAP envelope with the following structure:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <ord:getDistrOrderRate>
      <ord:request>
        <ordl:ChannelTypeCode?</ordl:ChannelTypeCode>
        <ordl:MailingDate?</ordl:MailingDate>
        <ordl:PoType?</ordl:PoType>
        <ordl:CustomerNumber?</ordl:CustomerNumber>
        <ordl:ContractNumber?</ordl:ContractNumber>
        <ordl:OutletId?</ordl:OutletId>
        <ordl:ContinuousInboundFreight?</ordl:ContinuousInboundFreight>
        <ordl:OrderOption>
          <ordl:Code?</ordl:Code>
          <ordl:Qualifier?</ordl:Qualifier>
        </ordl:OrderOption>
        <ordl:Item>
          <ordl:PriceLevel?</ordl:PriceLevel>
          <ordl:Origin>
            <ordl:Fsa?</ordl:Fsa>
            <ordl:Ldu?</ordl:Ldu>
          </ordl:Origin>
          <ordl:Destination>
  
```


#2: Test Using Large Amounts of Realistic Data

Test Using Large Amounts of Realistic Data

- Problems
 - The vast majority of service tests run with fixed test data
 - Much of this data is hard-coded
 - This doesn't truly exercise the back-end logic
 - *e.g. query optimizers*
 - Many developers and testers don't interact with business users
 - This makes it hard to understand what to feed the tests
 - Information returned from services often isn't tracked

Test Using Large Amounts of Realistic Data

- Implications
 - Actual performance metrics are unknown
 - Business logic isn't fully examined
 - Corner condition and boundary checks aren't performed
 - SLA compliance may be different in production

Test Using Large Amounts of Realistic Data

- Best Practices
 - Make sure that your testing tool allows for dynamic data
 - Leverage the power of a relational database
 - For input data
 - For output results
 - Use a data generator
 - Very inexpensive tools exist to fill a database with meaningful test data
 - This data can then be fed into your tests

#3: Make Sure Your Services Are Secure

Make Sure Your Services are Secure

- Problems
 - It's difficult to test security when the services haven't yet been written
 - Tests for back-end attacks such as SQL injections are often skipped
 - Access control is rarely tested properly

Make Sure Your Services are Secure

- Implications
 - You're unprepared for a potential attack
 - Retrofitting your services (and tests) for security will be difficult and expensive
 - There's no guarantee that your retrofitting will be effective

Make Sure Your Services are Secure

- Best Practices
 - Plan for security tests up-front: don't wait until the last minute
 - Set up realistic mock services (including security), and write tests against them
 - Make use of a broad spectrum of security assertion tests
 - Perform boundary and corner security tests

Sample Security Scenarios

- Analysis vectors
 - Authentication
 - Authorization
 - Integrity
 - Privacy/Confidentiality
 - Availability
 - Logging

Sample Security Scenarios

- Attack vectors
 - Parameter Tampering
 - Injection (SQL/XPath/LDAP)
 - Denial of Service / Distributed Denial of Service
 - Replay
 - WSDL Spoofing
 - XML Poisoning

#4: Get the Most Productivity From Your Developers and Testers

Get the Most Productivity From Your Developers and Testers

- Problems
 - Poor communication between developers and testers
 - Tests aren't part of software build process
 - Hand-coding of tests
 - Only basic tests get performed
 - Users are removed from testing their services
 - Often because of a lack of a user interface
 - Difficult to evaluate complex XML documents

Get the Most Productivity From Your Developers and Testers

- Implications
 - Tests are performed less frequently
 - Developers don't learn of problems until late in the cycle
 - Overall testing effort is disjointed and incomplete
 - Agile/SCRUM efforts are jeopardized
 - Failure-prone services are placed into production

Get the Most Productivity From Your Developers and Testers

- Best Practices
 - Bring testing all the way into the automated build process
 - Via integration with ant & maven
 - Let end users perform tests via a forms-based UI
 - Add libraries of pre-packaged tests and logic
 - Make sure to apply assertions to complex messages

Assertion Management

The screenshot displays a testing tool interface with the following components:

- Navigator:** Shows a project tree with 'PostComplaint Test Case' selected.
- Request/Response Panel:** Shows a SOAP response from 'http://MAVEN:8088/mockComplaintSOAPBinding'. The response body contains a `wsdl:ComplaintAcknowledge` element with the following details:

<code>wsdl:ShipmentIDAck</code>	987654321
<code>wsdl:ComplaintReceivedDate</code>	Wed Oct 28 10:44:23 PDT 2009
<code>wsdl:ComplaintStatus</code>	FAIL
<code>wsdl:ComplaintMessage</code>	Shipment ID not found
- Assertion Results:**
 - Not SOAP Fault - VALID
 - SOAP Response - VALID
 - Contains - FAILED
 - > Missing token [OK] in Response
 - Response SLA - FAILED
 - > Response did not meet SLA 771/200
- Summary:** 4 Assertions (4) failed, 21 in Request Log. Response time: 771ms (576 bytes).

#5: Fully Track Your Test Results

Fully Track Your Test Results

- Problems
 - Test cases often exist separately from business-driven requirements
 - Actual test cases are often a fraction of what should have been run
 - Test results are frequently not tracked
 - Even when tracked, reporting is often haphazard

Fully Track Your Test Results

- Implications
 - QA teams may have false sense of security
 - Management has no visibility into QA process
 - Resources may be incorrectly assigned

Fully Track Your Test Results

- Best Practices
 - Set up a formalized reporting mechanism
 - Review your results on a regular basis
 - Leverage graphical and business intelligence tools
 - Take action on what you learn

Reporting Example

Result Metrics

Result Metrics

🕒 Start Time	Fri Sep 25 21:11:05 PDT 2009
🕒 End Time	Fri Sep 25 21:11:05 PDT 2009
🕒 Time Taken	150 ms
✅ TestCase Count	5
❌ Failed TestCase Count	2
📋 TestStep Count	5
❌ Failed TestStep Count	2
🔴 Assertion Count	5
🔴 Failed Assertion Count	2

TestCase Results

TestCase	Status	Start Time	Time Taken	Reason
Contains Correct Status	FAILED	21:11:05	98 ms	Cancelling due to failed test step
Schema Compliance	FINISH	21:11:05	104 ms	
SLA Acceptable TestCase	FAILED	21:11:05	116 ms	Cancelling due to failed test step
No SOAP Fault TestCase	FINISH	21:11:05	122 ms	
Contains Correct Shipment	FINISH	21:11:05	122 ms	

Contains Correct Status TestCase Summary

Status	Start Time	Time Taken	Reason
FAILED	21:11:05	98 ms	Cancelling due to failed test step

#6: Test Your Services Under Anticipated Loads

Test Your Services Under Anticipated Loads

- Problems
 - Tests are rarely placed under load
 - Even when load tests are done, they don't match the expected real-world usage
 - “Load tests” is a blanket term. As we'll see, there are several types of these tests.
 - Long-running load tests often erroneously fail because of a single time-out or other error
 - This often scuttles an entire load test strategy

Test Your Services Under Anticipated Loads

- Implications
 - Service performance might be erratic
 - Especially if compositions are part of production landscape
 - Spotty responsiveness might lead to service duplication
 - SLA commitments might be impossible to uphold

Test Your Services Under Anticipated Loads

- Best Practices
 - Don't shortchange this important step
 - Try to match your load tests with what you expect to see in the real world
 - Configure some flexibility into your load tests
 - Don't abort just because of one timeout or other minor error
 - Make use of different styles of load test
 - Functional
 - Behavioral
 - Performance
 - Requirements-driven

#7: Make Sure You Govern Your Services

Make Sure You Govern Your Services

- Problems
 - Services change all the time
 - Tests have a hard time keeping up
 - Users of a service rely on the service complying with its schema
 - This doesn't happen in many cases, though
 - Governance automation often is primitive, if it even exists

Make Sure You Govern Your Services

- Implications
 - Testing might not uncover compatibility issues
 - Services might become unusable because of contract changes
 - Failures might lead to service proliferation
 - Failures might lead to abandoning the SOA initiative

Make Sure You Govern Your Services

- Best Practices
 - Ensure that services conform to standards
 - Ensure that services conform to their schema
 - Refactor your tests when contracts change
 - Invest in governance technology, even if rudimentary

Refactoring Tests

The image displays two screenshots of an IDE's refactoring tool, likely SoapUI, used for refactoring tests.

Top Screenshot: Transfer Operations
 This window is titled "Refactor Definition" and "Transfer Operations". It is used to connect old and new operations. The interface is split into two columns: "Old Schema" and "New Schema".
 - **Old Schema Operations:** getCommOrderRate, getCommOrderServiceStandard, getDistrOrderRate.
 - **New Schema Operations:** getCommOrderServiceStandard, getDistrOrderRate, getCommOrderRate1.
 Arrows indicate the mapping: getCommOrderRate maps to getCommOrderServiceStandard, getCommOrderServiceStandard maps to getDistrOrderRate, and getDistrOrderRate maps to getCommOrderRate1.

Bottom Screenshot: Refactor Schema
 This window is also titled "Refactor Definition" and "Refactor Schema". It is used to map nodes in the old schema to the new schema. The interface is split into two columns: "Old Schema" and "New Schema".
 - **Old Schema Nodes:** getCommOrderRate Request, Body, getCommOrderRate, request, PoType, Header.
 - **New Schema Nodes:** getCommOrderRate1 Request, Body, getCommOrderRate, request, CustomerAge, Temperature, PoType (not in schema), Header (not in schema).
 Arrows indicate the mapping: getCommOrderRate Request maps to getCommOrderRate1 Request; Body maps to Body; getCommOrderRate maps to getCommOrderRate; request maps to request; PoType maps to PoType (not in schema); Header maps to Header (not in schema).
 At the bottom, there are buttons for Filter, Connect, Disconnect, Discard, Set Value, Pretty Print, Clear Error, Edit Manually, and Save.

Seven Fundamentals of Mission-Critical Service Testing

Robert D. Schneider
November 20, 2009

robert.schneider@think88.com